Learning Multiplicity Tree Automata*,**

Amaury Habrard[1] and Jose Oncina[2,***]

[1] LIF – Université de Provence
39, rue Frédéric Joliot Curie – 13453 Marseille cedex 13 – France
amaury.habrard@lif.univ-mrs.fr
[2] Dep. de Lenguajes y Sistemas Informático
Universidad de Alicante E-03071 Alicante – Spain
oncina@dlsi.ua.es

Abstract. In this paper, we present a theoretical approach for the problem of learning multiplicity tree automata. These automata allows one to define functions which compute a number for each tree. They can be seen as a strict generalization of stochastic tree automata since they allow to define functions over any field K. A multiplicity automaton admits a support which is a non deterministic automaton. From a grammatical inference point of view, this paper presents a contribution which is original due to the combination of two important aspects. This is the first time, as far as we now, that a learning method focuses on non deterministic tree automata which computes functions over a field. The algorithm proposed in this paper stands in Angluin's exact model where a learner is allowed to use membership and equivalence queries. We show that this algorithm is polynomial in time in function of the size of the representation.
Keywords: multiplicity tree automata, recognizable tree series, learning from equivalence and membership queries.

1 Introduction

Trees are natural candidates for modeling a hierarchy in data, and for example they are particularly relevant to model a web page. Recently, due to the potential applications in the web, a lot of machine learning approaches devoted to trees have been proposed. From a grammatical inference standpoint, the natural objects for dealing with tree-structured data are tree automata and tree languages [1, 2]. These objects are natural extensions of finite automata on strings, except that the alphabet is constituted of functional symbols representing labels of tree nodes. Several learning algorithms has been proposed in the literature for learning tree automata. Among them we can cite those of Knuutila *et al.* [3], Garcia *et*

* This work was supported in part by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778. This publication only reflects the authors' views.
** This work is part of the ARA marmota french projet.
*** This work was done when the second author was visiting the LIF Marseille.

al. [4] and Kosala *et al.* [5] for dealing with an unranked alphabet. Besombes *et al.* [6] have studied the learning of regular tree languages using positive examples and membership queries. Carme *et al.* [7] have proposed to learn specific tree transducers for information extraction applications. In the probabilistic case, we can cite Carrasco *et al.* [8] and Rico *et al.* [9]. An important remark has to be made here: In general, these approaches have focused on learning deterministic models which may imply the construction of models with a high complexity.

In this paper, we propose to study the learnability of multiplicity tree automata. Informally, a multiplicity tree automaton defines a function allowing one to associate to any tree a value in a field K (for example $\mathbb{R}$ or $\mathbb{Q}$). We call such an automaton a K-multiplicity tree automaton. For example, stochastic tree automata are then particular cases of multiplicity automata where $K = [0, 1]$. However, multiplicity automata do not compute stochastic distribution in general. For example, you can define multiplicity tree automata that represent a function which computes the number of occurrences of a given symbol in a tree. There exists a notion of support for a multiplicity automaton which corresponds to a non deterministic finite tree automaton. This non determinism characterize a greater expressiveness than deterministic models. Another crucial point has to be made here. For defining a multiplicity automaton, we define functions which compute the value of a subtree when a transition is applied to analyse the subtree. In the case of a multiplicity automaton these functions are *multilinear* (*e.g.* for symbol of arity p we define a $p-$linear function) and offer a good expressiveness for defining the global function computed by a multiplicity automaton. In fact, the function computed by multiplicity automata are usually referred as recognizable formal power series on trees [10, 11].

Hence, this article combines two main improvements for learning tree automata. The method we present allows us to learn tree automata with a non deterministic support that compute functions from a set of trees to a field K. This is, as far as we know, the first time that a learning method is proposed for multiplicity tree automata. We think that this research direction can be very promising due to the potential applications notably in information extraction from the web.

In the case of strings, some learning methods of multiplicity automata have already been published: [12]. But, there exists no similar work for trees. The adaptation to trees is not trivial since the use of multilinear mappings for transitions are a real improvement in comparison with the string case.

We propose a learning algorithm for multiplicity tree automata which stands in Angluin's *exact learning model* [13]. In this framework, the learning algorithm is allowed to ask queries to an oracle. The algorithm can ask equivalence queries to know if he found the correct hypothesis. In the opposite case, a counterexample is returned by the oracle. Membership queries are also available to have information about one example. The algorithm we present runs in polynomial time in the size of the target and needs a number of queries also polynomial in the size of the target. This algorithm exploits a result showing that the number of states of a minimal multiplicity tree automaton is exactly the rank (which is

finite) of the Hankel matrix of the power tree series it represents. The underlying principle of our approach is to find a hierarchical basis which allows to generate all the element of the target multiplicity automaton. This basis is then used for building the tree automaton which is minimal since a basis is a minimal representation by definition.

The paper is organised as follows. In Section 2, we introduce some background about recognizable formal tree series and multiplicity automata. The Hankel Matrix associated to a recognizable series is defined in Section 3. In this section, we also characterize recognizable tree series in terms of the dimension of their associated Hankel Matrix. The learning algorithm is presented in Section 4.

2 Background

To begin with, we introduce the alphabet for defining trees and the concept of free magma which is an equivalent to the free monoid Σ^* over strings.

Following [10], Let F be a finite set of function symbols, that is a ranked alphabet $F = F_0 \cup F_1 \cup \cdots \cup F_p$. The elements in F_p are the function symbols of *arity* p. We denote by $M(F)$ the free magma generated by F. The elements in $M(F)$ are called *trees*. If t is a tree and $t \notin F_0$ then there exists an integer $p \geq 0$, a symbol function $f \in F_p$, and trees $t_1, \ldots, t_p$ such that $t = f(t_1, \ldots, t_p)$.

Definition 1. *Let K be a commutative field. A* formal power tree series (TS) *on $M(F)$ with coefficients in K is a mapping*

$$S : M(F) \to K$$

The set of all TS on $M(F)$ with coefficients in K is denoted by $K\{\{F\}\}$.

2.1 Recognizable Formal Power Tree Series (RTS)

Let V be a finite dimensional vector space over the field K, let $dim(V)$ be the dimension of V and let $x \in V$, we denote by $[x]_i$ the ith coordinate of the vector x. In the following, the vector V will represent intermediate values used in a non deterministic analysis of a tree by a multiplicity automaton. Each dimension of V will correspond to the result associated to a state of the multiplicity tree automaton.

We denote by $\mathcal{L}(V^p; V)$ the set of p-linear mappings from V^p to V. Let $\mathcal{L} = \cup_{p \geq 0} \mathcal{L}(V^p; V)$. The vector space V is an $\mathcal{L}$-magma with $\mathcal{L}(V^p; V)$ as the function set with arity p. Thus any mapping $\mu : F \to \mathcal{L}$ which maps F_p into $\mathcal{L}(V^p; V)$ converts V into an F-magma.

Intuitively, μ will correspond to a transition in the automaton that defines the next states used during the analysis of a tree.

Definition 2. A linear representation *of the free magma $M(F)$ is a couple (V, μ), where V is a finite dimensional vector space over K, and where $\mu : F \to \mathcal{L}$ maps F_p into $\mathcal{L}(V^p; V)$ for each $p \geq 0$.*

Thus for each $f \in F_p$, $\mu(f) : V^p \to V$ is p-linear, and since $M(F)$ is free, μ extends uniquely to a morphism $\mu : M(F) \to V$ by the formula

$$\mu(f(t_1, \ldots, t_p)) = \mu(f)(\mu(t_1), \ldots, \mu(t_p)). \quad (1)$$

Definition 3. *let S be a TS on $M(F)$, then S is a* recognizable TS *(RTS) if there exists a triple (V, μ, λ), where (V, μ) is a linear representation of $M(F)$, and $\lambda : V \to K$ is a linear form, such that $S(t) = \lambda(\mu(t))$ for all t in $M(F)$.*

(V, μ, λ) is called a *Multiplicity Tree Automaton* (MTA) and we say that (V, μ, λ) is a representation of S. Intuitively, if we try to make the link between these automata and classical automata in language theory, the states of a multiplicity automaton correspond to a basis of V. The transitions are defined by μ and the final states by λ. According to this definition, multiplicity tree automata can be seen as an extension of bottom-up tree automata.

In order to illustrate these objects, we provide an example introduced in [10]. We consider a recognizable tree series S (*i.e.* a multiplicity tree automaton) computing the number of occurrences of a symbol f in a tree.

We define a MTA (V, λ, μ) where $V = \mathbb{Q}^2$ and (e_1, e_2) a canonical basis of V (*i.e.* $e_1 = (1, 0)$ and $e_2 = (1, 0)$). We define μ and λ such that:

$$\forall g \in F_q, g \neq f : \mu(g)(e_{i_1}, \ldots, e_{i_q}) = \begin{cases} e_1 \text{ if } e_{i_1} = \cdots = e_{i_q} = e_1 \\ e_2 \text{ if there exists exactly one } e_{i_j} \text{ s.t. } e_{i_j} = e_2 \\ 0 \text{ otherwise} \end{cases}$$

$$f \in F_p : \mu(f)(e_{i_1}, \ldots, e_{i_p}) = \begin{cases} e_1 + e_2 \text{ if } e_{i_1} = \cdots = e_{i_p} = e_1 \\ e_2 \text{ if there exists exactly one } e_{i_j} \text{ s.t. } e_{i_j} = e_2 \\ 0 \text{ otherwise} \end{cases}$$

$$\forall a \in F_0 : \mu(a) = \begin{cases} e_1 \text{ if } \ a \neq f \\ e_1 + e_2 \text{ if } \ a = f \in F_0 \\ 0 \text{ otherwise} \end{cases}$$

Finally $\lambda(e_1) = 0$ and $\lambda(e_2) = 1$.

It can be shown that $\mu(t) = e_1 + S(t)e_2$, then $\lambda(\mu(t)) = S(t)$. Let's see an example of a computation of $S(t)$. Consider $t = f(a, g(f(a, a)))$ over the ranked alphabet $F_0 = \{a\}$, $F_1 = \{g(\cdot)\}$ $F_2 = \{'f(\cdot, \cdot)'\}$.

$$\begin{aligned} \mu(f(a, g(f(a, a)))) &= \mu(f)(\mu(a), \mu(g(f(a, a)))) \\ &= \mu(f)(e_1, \mu(g)(\mu(f(a, a)))) \\ &= \mu(f)(e_1, \mu(g)(\mu(f)(\mu(a), \mu(a)))) \\ &= \mu(f)(e_1, \mu(g)(\mu(f)(e_1, e_1))) \\ &= \mu(f)(e_1, \mu(g)(e_1 + e_2)) \\ &= \mu(f)(e_1, \mu(g)(e_1)) + \mu(f)(e_1, \mu(g)(e_2)) \\ &= \mu(f)(e_1, \mu(g)(e_1)) + \mu(f)(e_1, \mu(g)(e_2)) \\ &= \mu(f)(e_1, e_1) + \mu(f)(e_1, e_2) = e_1 + e_2 + e_2 = e_1 + 2e_2 \end{aligned}$$

Hence $\lambda(\mu(t)) = 2$.

2.2 Contexts

We introduce contexts which allow us to define an equivalent notion of concatenation for trees. Let \$ be a zero arity function symbol not in F_0, a context is an element of the free-magma $M(F \cup \{\$\})$ such that the symbol \$ appears exactly one time. We denote by $M(\$, F)$ such set.

Let $c \neq \$$ be a context, then, there exists two integers p and n ($n \leq p$), a symbol function $f \in F_p$, trees $t_1, \ldots, t_{p-1}$ and a context c' such that $c = f(t_1, \ldots, t_{n-1}, c', t_n, \ldots, t_{p-1})$. Let t be a tree and let c be a context, $t \cdot c$ denotes the tree obtained by substituting the symbol \$ in the context c by the tree t.

The μ function can be extended to work over contexts ($\mu : M(\$, F) \to \mathcal{L}(V; V)$) recursively on the following way:

$$\mu(\$)(x) = x$$
$$\mu(f(t_1, \ldots, t_{n-1}, c, t_n, \ldots, t_p))(x) = \mu(f)(\mu(t_1), \ldots, \mu(t_{n-1}), \mu(c)(x), \mu(t_n), \ldots, \mu(t_p))$$

It is easy to see that $\mu(t \cdot c) = \mu(c)(\mu(t))$. Let t be a tree we define $\mathrm{Suf}(t) = \{c' : \exists t', t' \cdot c = t\}$.

2.3 Multilinear Functions

Let $V = K^d$, let $f : V^p \to V$ be a p-linear function, such that $f(x_1, \ldots x_p) = y$ then f can be expressed as:

$$[y]_i = \sum_{\substack{1 \leq j_i \leq d \\ i=1,\ldots,p}} f_{i,j_1,\ldots,j_p} [x_1]_{j_1} \ldots [x_p]_{j_p} \tag{2}$$

where $f_{i,j_1,\ldots,j_p}$ are the d^{p+1} parameters that define the function f. Note that in order to fully specify an MTA (V, μ, λ) ($V = K^r$), we need:

- d parameters to specify λ
- $\sum_{i:|F_i|\neq 0} |F_i| d^{i+1}$ parameters to specify the multilinear functions.

Let (V, μ, λ) a MTA, as $\lambda : V \to K$ is a linear function it can be represented as a vector ($\lambda = (\lambda_1, \ldots, \lambda_d)$).

In the same way, when applied over contexts, μ gives a linear function over vectors in V ($\mu : M(\$, F) \to \mathcal{L}(V; V)$), this linear function can be represented as a $d \times d$ matrix. For example, Using this notation, the equation $\lambda(\mu(t \cdot c)) = \lambda(\mu(c)(\mu(t)))$ can be written clearly as $\mu(t \cdot c)\lambda = \mu(t)\mu(c)\lambda$.

3 Hankel Matrix and Recognizable Tree Series

3.1 The Hankel Matrix

Informally, the Hankel Matrix of a TS S is an infinite matrix that represents all the possible values for S. The rows of the matrix are indexed by trees and the columns by contexts. A value in this matrix corresponds to the value in the series S for the tree built by the concatenation of the tree on the row and the context on the column. An example is drawn on Figure 1.

Definition 4. *The* Hankel matrix *(HM) of a TS $S \in K\{\{F\}\}$ (HM(S)) is an infinite matrix $H : M(F) \times M(\$, F) \to K$ such that $H_{t,c} = S(t \cdot c)$ $\forall t \in M(F), c \in M(\$, F)$.*

	$\$$	$f(\$, a)$	$f(\$, b)$	$f(a, \$)$	$\cdots$
a	$S(a)$	$S(f(a,a))$	$S(f(a,b))$	$S(f(a,a))$	$\cdots$
b	$S(b)$	$S(f(b,a))$	$S(f(b,b))$	$S(f(a,b))$	$\cdots$
$f(a,a)$	$S(f(a,a))$	$S(f(f(a,a),a))$	$S(f(f(a,a),b))$	$S(f(a,f(a,a)))$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Fig. 1: An example of the Hankel Matrix of a tree series S.

Let t be a tree, a row of the Hankel Matrix is defined by H_t, then $H_t(c) = H_{t,c}$.

Lemma 1. *Let (V, μ, λ) an MTA of a TS $S \in K\{\{F\}\}$, let H the Hankel Matrix of S, then $rank(H) \leq dim(V)$.*

Proof. Let $d = dim(V)$, define a $\infty \times d$ matrix R such that $R(t, i) = [\mu(t)]_i$ and define a $d \times \infty$ matrix C such that $C(i, c) = [\mu(c)\lambda]_i$. Clearly, $H = RC$, but as $rank(C) \leq d$ and $rank(R) \leq d$, then $rank(H) \leq d$. □

3.2 Recognizable Tree Series

We are going to show that a TS is recognizable if an only if its HM has a finite rank. For this purpose, we will introduce the notion of hierarchical generator and show that this generator allows us to generate all the entries of the HM.

Definition 5. *Let H be an HM, we say that $B = \{e_1, \ldots, e_d\}$ is a* hierarchical generator *(HG) in H if*

1. *each $e_i \in B$ is* linearly independent *(LI) of the rest ($\nexists \alpha_j \in K : H_{e_i} = \sum_{1 \leq j \leq d,\, i \neq j} \alpha_i H_{e_j}$)*
2. *$\forall e_i = f(e_{i_1}, \ldots, e_{i_p}) \in B$ then $e_{i_j} \in B$, $j = 1, \ldots, p$.*
3. *There is no tree $t = f(e_{i_1}, \ldots, e_{i_p})$, $t \notin B$, $e_{i_j} \in B, j = 1, \ldots, d$, such that t is LI of B.*

Algorithm HG(H)

input: Hankel matrix H
output: a hierarchical generator B

1. initialize $B = \{\}$, $i = 1$
2. search for the smaller tree $e_i = f(e_{i_1}, \ldots, e_{i_p})$ such that $e_{i_j} \in B$, $j = 1, \ldots, p$ and H_{e_i} is not a linear combination of $\{H_{e_1}, \ldots, H_{e_{i-1}}\}$.
3. if not found then halt.
   if found then add it to B and increment i.
4. go to 2

Algo. 1: Algorithm HG to obtain a Hierarchical Generator from a Hankel Matrix.

Algorithm MTAB(H)

input: a Hankel matrix H
output: a representation (V, μ, λ)

1. Obtain a hierarchical generator $e_1, \ldots, e_d$ from H.
2. Set $dim(V) = d$.
3. Set $\lambda_i = H(e_i)$, $i = 1, \ldots, d$
4. $\forall p, \forall f \in F_p, \forall i_1, \ldots, i_p$, $1 \leq i_j \leq d$, let α_i such that: $H_{f(e_{i_1}, \ldots, e_{i_p})} = \sum_{i=1}^{r} \alpha_i H_{e_i}$. Then fix $f_{i,i_1,\ldots,i_p} = \alpha_i$.

Algo. 2: Algorithm MTAB that builds representation from a Hankel Matrix.

Given any HM H, note that the previous definition does not imply that any HG for H should be a basis for the HM. A HG can be easily extracted from an HM following Algorithm 1 HG. In order to eliminate any ambiguity, we should fix an arbitrary order on trees, any order can be chosen.

Algorithm 2 describes the algorithm MTAB to extract an MTA from an HM. Although in the present version it works on infinite HM (then it is not really an algorithm) we are going to use it later with finite size portions of the HM.

Now we present a series of lemma necessary to show that the multiplicity automaton extracted by algorithm MTAB represents the TS defined by its HM. Let δ_{ij} representing a function that returns always 0 except when $i = j$ it returns 1.

Lemma 2. *Let H be the Hankel matrix of a RTS $S \in \{\{K\}\}$. Let $(V, \mu, \lambda) =$ MTAB(H), then*

$$[\mu(e_i)]_j = \delta_{ij}.$$

Proof. Note that as e_i are elements of the generator and let $e_i = f(e_{i_1}, \ldots, e_{i_p})$, using the algorithm, $f_{k,i_1,\ldots,i_p} = \delta_{ik}$.

We prove the result by induction in the height of the trees in the generator. Let $e_i \in F_0$, then $p = 0$ and $f_k = \delta_{ik}$, in such case, $[\mu(e_i)]_j = f_j = \delta_{ij}$ as required.

Now let $e_i = f(e_{i_1}, \ldots, e_{i_p})$ and suppose, by induction, that $[\mu(e_{i_j})]_k = \delta_{i_j k},\ j = 1, \ldots, p$. Then,

$$\begin{aligned}
[\mu(e_i)]_j &= [\mu(f(e_{i_1}, \ldots, e_{i_p}))]_j \\
&= [\mu(f)(\mu(e_{i_1}), \ldots, \mu(e_{i_p}))]_j && \text{by Equation 1} \\
&= \sum_{j_l} f_{j,j_1,\ldots,j_p} [\mu(e_{i_1})]_{j_1} \ldots [\mu(e_{i_p})]_{j_p} && \mu(f) \text{ is a multilinear function} \\
&= \sum_{j_l} f_{j,j_1,\ldots,j_p} \delta_{i_1 j_1} \ldots \delta_{i_p j_p} && \text{induction step} \\
&= f_{j,i_1,\ldots,i_p} = \delta_{ij} && \text{by Algorithm } \texttt{MTAB} \quad \square
\end{aligned}$$

Note that as a consequence of the Lemma 2, $\mu(e_i)\lambda = H(e_i), i = 1, \ldots, r$.

Lemma 3. *Let H be the HM of a RTS $S \in \{\{K\}\}$. Let $(V, \mu, \lambda) = \texttt{MTAB}(H)$. Let $B = \{e_1, \ldots, e_d\} = \texttt{HG}(H)$. then,*

$$\mu(f(e_{j_1}, \ldots, e_{j_p})) = \sum_{j=1}^{r} f_{j,j_1,\ldots,j_p} \mu(e_j). \tag{3}$$

Proof.

$$\begin{aligned}
\left[\mu(f(e_{j_1}, \ldots, e_{j_p}))\right]_i &= \left[(\mu f)(\mu(e_{j_1}), \ldots \mu(e_{j_p}))\right]_i && \text{by Equation 1} \\
&= \sum_{1 \le k_i \le p} f_{i,k_1,\ldots,k_p} [\mu(e_{j_1})_{k_1}] \ldots [\mu(e_{j_p})_{k_p}] && \text{by Equation 2} \\
&= \sum_{1 \le k_i \le p} f_{i,k_1,\ldots,k_p} \delta_{j_1,k_1} \ldots \delta_{j_p,k_p} && \text{by Lemma 2} \\
&= f_{i,j_1,\ldots,j_p} \\
&= \left[\sum_{j=1}^{r} f_{j,j_1,\ldots,j_p} \delta_{ij}\right]_i = \left[\sum_{j=1}^{r} f_{j,j_1,\ldots,j_p} \mu(e_j)\right]_i && \text{by Lemma 2} \quad \square
\end{aligned}$$

Lemma 4. *Let H be the HM of a RTS $S \in \{\{K\}\}$. Let $(V, \mu, \lambda) = \texttt{MTAB}(H)$. Let $B = \{e_1, \ldots, e_d\} = \texttt{HG}(H)$. Then, for all context c*

$$H_{e_i}(c) = \mu(e_i)\mu(c)\lambda.$$

Proof. By induction, the base case ($c = \$$) is evident since $\mu(e_i)\lambda = H_{e_i}(\$)$. Let us show that $H_{f(e_{j_1},\ldots,e_{j_p})}(c) = \mu(f(e_{j_1}, \ldots, e_{j_p}) \cdot c)\lambda$.

$$
\begin{aligned}
H_{f(e_{j_1},\ldots,e_{j_p})}(c) &= \sum_{j=1}^{d} f_{j,j_1,\ldots,j_p} H_{e_j}(c) && \text{by algorithm } \texttt{MTAB} \\
&= \sum_{j=1}^{d} f_{j,j_1,\ldots,j_p} \mu(e_j)\mu(c)\lambda && \text{by induction step} \\
&= \mu(f(e_{j_1},\ldots,e_{j_p}))\mu(c)\lambda && \text{by Lemma 3} \\
&= \mu(f(e_{j_1},\ldots,e_{j_p})\cdot c)\lambda && \square
\end{aligned}
$$

Theorem 1. *Let H be the HM of a RTS $S \in \{\{K\}\}$. Let $(V,\mu,\lambda) = \texttt{MTAB}(H)$. Then, $S(t) = \mu(t)\lambda$.*

Proof. Let $B = \{e_1,\ldots,e_d\} = \texttt{HG}(H)$. Note that for $f \in F_0$

$$
\begin{aligned}
H_f(c) &= \sum_{i=1}^{d} f_i H_{e_i}(c) && \text{by Algorithm } \texttt{MTAB} \\
&= \sum_{i=1}^{d} f_i \mu(e_i)\mu(c)\lambda && \text{by Lemma 4} \\
&= \sum_{i=1}^{d} f_i \sum_{j=1}^{d} [\mu(e_i)]_j \, [\mu(c)\lambda]_j \\
&= \sum_{i=1}^{d} f_i \sum_{j=1}^{d} \delta_{ij} \, [\mu(c)\lambda]_j && \text{by Lemma 2} \\
&= \sum_{i=1}^{d} f_i \, [\mu(c)\lambda]_i \\
&= \sum_{i=1}^{d} \mu(f)_i \, [\mu(c)\lambda]_i && \text{by Equation 2} \\
&= \mu(f)\mu(c)\lambda = \mu(f\cdot c)\lambda
\end{aligned}
$$

And we have what we wanted because for any tree t, it exists an $f \in F_0$ and a context c such that $t = a \cdot c$. Then, $S(t) = H_f(c) = \mu(f\cdot c)\lambda = \mu(t)\lambda$. $\square$

Corollary 1. *Let H be the Hankel Matrix of a RTS $S \in \{\{K\}\}$. Then, the MTA (V,μ,λ) that represents S with a smaller $dim(V)$ satisfies that $rank(H) = dim(V)$.*

Proof. By Lemma 1 we have that $rank(H) \leq dim(V)$, and by Theorem 1 we have shown that, using algorithm `MTAB`, we can build a representation (V,μ,λ) consistent with H such that $dim(V) \leq rank(H)$. $\square$

Corollary 2. *Let H be the HM of a RTS $S \in \{\{K\}\}$, $\texttt{HG}(H)$ is a basis, i.e. $\texttt{HG}(H)$ can generate all the Hankel Matrix H.*

Corollary 3. *Let H be the Hankel Matrix of a RTS $S \in \{\{K\}\}$. The MTA $(V, \mu, \lambda) = \texttt{MTAB}(H)$ is the representation of S that minimizes $dim(V)$.*

4 Inference algorithm

The learning model we use is the exact learning model of Angluin [13]. Let f be a target function. At each step of the inference procedure, the learning algorithm can propose an hypothesis function h by making an equivalence query (EQ) to an oracle. This oracle answers YES if h is equivalent to f on all input assignments. In this case the target is identified, the learning algorithm succeeds and halts. Otherwise, the answer to the equivalence query is NO and the algorithm receives a counterexample, that is an assignment z such that $f(z) \neq h(z)$. Moreover, the learning algorithm is also allowed to query an oracle for the value of the function f on a particular assignment z by making a membership query (MQ) on z. The response to such a query is the value $f(z)$. We say that the learner identifies a class of functions $\mathcal{F}$, if, for every function $f \in \mathcal{F}$, the learner outputs an hypothesis h that is equivalent to f and does so in polynomial time in the "size" of a shortest representation of f and the length of the longest counterexample.

To begin with, we define an experimental table which corresponds to a submatrix of the Hankel Matrix of a the target series S.

Definition 6. *An* experiment table *(ET) is a 3-tuple $(T, C, \hat{H})$ such that: T is a set of trees, C is a set of contexts, $\hat{H} : T \times C \to K$ a submatrix of an HM.*

In the following, we will maintain this table filled such that $\hat{H}_{t,c} = S(t \cdot c)$.

Next definition will allows us to apply algorithm `MTAB` to any Experiment Table.

Definition 7. *Let $M = (T, C, \hat{H})$ be an ET and let $B = \texttt{HG}(\hat{H})$. M is* closed *if $\forall p, \forall f \in F_p, \forall e_{i_1}, \ldots, e_{i_p} \in B$, $f(e_{i_1}, \ldots, e_{i_p}) \in T$*

Algorithm 3 `close` is able to close any ET. Note that any call to `close` will include all the symbols in F_0.

The following definition will ensure that any extracted MTA by `MTAB` is consistent with the data in the Experiment Table.

Definition 8. *Let $M = (T, C, \hat{H})$ be a closed ET, let $(V, \mu, \lambda) = \texttt{MTAB}(M)$. An ET is* consistent *if $\forall t \in T, \forall c \in C, \mu(t \cdot c)\lambda = \hat{H}_{t,c}$.*

Definition 9. *A set of contexts C is suffix complete if $\forall c \in C, \forall c_1, c_2 : c_1 \cdot c_2 = c$ then $c_2 \in C$.*

Lemma 5. *Let $M = (T, C, \hat{H})$ be a closed ET such that C is suffix complete and let $(V, \mu, \lambda) = \texttt{MTAB}(M)$. For any tree $t = f \cdot c : f \in F_0, c \in C$, then $\mu(f \cdot c)\lambda = \hat{H}_{f,c}$.*

Algorithm `close`(M)

input: an ET $M = (T, C, \hat{H})$
output: a closed ET M

1. Let $B \leftarrow \texttt{HG}(M)$
2. if $\exists p, \exists f \in F_p, \exists e_{i_1}, \ldots, e_{i_p} \in B$ $f(e_{i_1}, \ldots, e_{i_p}) \notin T$ then $T \leftarrow T \cup f(e_{i_1}, \ldots, e_{i_p})$ else halt
3. go to 1

Algo. 3: Algorithm `close` allowing to close a table.

Algorithm `consistent`(M)

input: an experiment table $M = (T, C, \hat{H})$
output: a consistent table M

1. $M \leftarrow \texttt{close}(M)$
2. if exists $t \in T, c \in C : \hat{H}_{t,c} \neq \mu(t \cdot c)\lambda$ then $C \leftarrow C \cup \mathrm{Suf}(t \cdot c)$ else halt
3. make membership queries to fill $\hat{H}$
4. go to 1

Algo. 4: Algorithm `consistent` allowing to keep a table consistent

Proof. Sketch. A similar technique used in the proof of Theorem 1 allows to prove the result. Note that since some proofs of lemmas needed to show Theorem 1 use induction over trees and contexts, C should be suffix complete in order to guarantee its correctness. □

Lemma 6. *Let $M = (C, T, \hat{H})$ be a closed table with C suffix complete and let $(V, \mu, \lambda) = \texttt{MTAB}(M)$. Consider $t \in T$ and $c \in C$ such that $\mu(t \cdot c)\lambda \neq \hat{H}_{t,c}$, we can decompose $t \cdot c = f \cdot c'$ where $f \in F_0$. Let $M' = (T', C', \hat{H}') = \texttt{closed}((T, C \cup \mathrm{Suf}(f \cdot c'), \hat{H}))$; then $\mu'(f \cdot c')\lambda = \hat{H'}_{f,c'}$. Moreover, $rank(\hat{H}') > rank(\hat{H})$.*

Proof. Sketch. A direct application of Lemma 5 ensures that $\mu'(f \cdot c')\lambda = \hat{H'}_{f,c'}$.

On the other hand, all the trees of the HG(M) are linearly independent. By adding new contexts in C, those trees remain linearly independent and then they will be a part of HG.

Let $A = \texttt{MTAB}(M)$ and $A' = \texttt{MTAB}(M')$. Clearly all the trees in $\texttt{HG}(M)$ are also in $\texttt{HG}(M')$ but new trees should appear in $\texttt{HG}(M')$, otherwise, by construction, $A = A'$ and this is impossible because $\mu(f \cdot c')\lambda \neq \mu'(f \cdot c')\lambda$.

Thus, $rank(\hat{H}') > rank(\hat{H})$. □

Theorem 2. *Let MQ and EQ respectively membership and equivalence oracles of a RTS S with an associated Hankel matrix H. Let $r = rank(H)$* `LearnMTA`(MQ, EQ) *returns the minimal representation compatible with the target in polynomial time making at most r equivalence queries and $|A|m$ membership queries, where m is the length of the longest counterexample returned by the Equivalence Queries.*

Proof. In the same way as Lemma 6, it can be shown that the counterexample acts in a similar way as when a non consistency is found in the ET. In any case, after the `consistent` call (after step 4 of `LearnMTA`) a new MTA compatible with all the data in the ET is obtained. As shown in Lemma 6, the hierarchical

Algorithm `LearnMTA`(EQ, MS)

input: an equivalence oracle EQ
input: a membership oracle MS
output: a MTA A

1. Initialize: $T = \{\}$, $C = \{\}$, $A =$ an empty MTA, $M = (T, C, \hat{H})$ an empty ET.
2. Ask an equivalence query $EQ(A)$.
   If the answer is YES then halt with output A.
   Otherwise the answer is NO and z is a counterexample.
3. Add Suf(z) to C.
4. $M \leftarrow$ `consistent`(M)
5. $A \leftarrow$ `MTAB`(M)
6. go to 2

Algo. 5: inference algorithm `LearnMTA`

generator HG of this MTA has strictly more trees than the previous one. As the number of trees in HG can not be bigger than the rank of the HM for the target MTA, then the process should finish and gives the correct hypothesis.

Now, looking at the time complexity, since all the steps of the algorithm can be done in polynomial time with respect to the size of the target MTA (linear independence of a set of vector, Algorithms `HG`, `MTAB`, `close` and `consistent`), it is evident that algorithm `LearnMTA` runs in polynomial time.

With respect to the queries, it is easy to see that the number of Equivalence Queries can no be larger than r (at most one for each tree added to HG).

Moreover, note that the all the results of the membership queries are stored in the equivalence table or were used to calculate the λ vector (of size r). It is easy to see that $|T| = \sum_{i:F_i \neq 0} |F_i| r^i$ and that $|C| \leq r.m$ since for a tree t we can define no more than $|t|$ contexts. Reminding that $|A| = \sum_{i:F_i \neq 0} |F_i| r^{i+1} + r$ then, the number of Membership Queries is lower that $|T||C| + r \leq |A|m$. □

5 Conclusion

In this paper we proposed a learning algorithm for identifying multiplicity tree automata that define functions associating a number to any tree. We showed that the size of a minimal multiplicity tree automaton is function of the finite rank of its Hankel matrix. Our algorithm is able to identify the minimal automata in polynomial time in the exact learning model of Angluin. The originality of our approach is to find a hierarchical basis that permits to generate all the elements of the target series.

We think that multiplicity tree automata can offer a wide range of potential applications, especially for extraction information from the web. We showed that this class of automata is identifiable and our perspective is to find efficient approaches in other learning paradigms. We have begin to study a possible ex-

tension of the approach presented in [14, 15], which allows to learn stochastic languages on strings represented by multiplicity automata, to trees.

References

1. Gécseg, F., Steinby, M.: Tree Automata. Akadémiai Kiadó, Budapest (1984)
2. Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications . Available from: `http://www.grappa.univ-lille3.fr/tata` (1997)
3. Knuutila, T., Steinby, M.: Inference of tree languages from a finite sample: an algebraic approach. Theoretical Computer Science **129**(2) (1994) 337–367
4. Garcia, P., Oncina, J.: Inference of recognizable tree sets. Research Report DSIC - II/47/93, Universidad Politécnica de Valencia (1993)
5. Kosala, R., Bruynooghe, M., den Bussche, J.V., Blockeel, H.: Information extraction from web documents based on local unranked tree automaton inference. In: Proceedings of IJCAI 2003. (2003) 403–408
6. Besombes, J., Marion, J.: Learning tree languages from positive examples and membership queries. In: Proceedings of ALT'04, Springer (2004) 440–453
7. Carme, J., Gilleron, R., Lemay, A., Niehren, J.: Interactive learning of node selecting tree transducer. Machine Learning (2006) to appear.
8. Carrasco, R., Oncina, J., Calera-Rubio, J.: Stochastic inference of regular tree languages. Machine Learning **44**(1/2) (2001) 185–197
9. Rico-Juan, J., Calera, J., Carrasco, R.: Probabilistic k-testable tree-languages. In: Proceedings of ICGI 2000. Volume 1891 of LNCS., Springer (2000) 221–228
10. Berstel, J., Reutenauer, C.: Recognizable formal power series on trees. Theoretical Computer Science **18** (1982) 115–148
11. Esik, Z., Kuich, W.: Formal tree series. Journal of Automata Languages and Combinatorics **8**(2) (2003) 219–285
12. Beimel, A., Bergadano, F., Bshouty, N., Kushilevitz, E., Varricchio, S.: Learning functions represented as multiplicity automata. Journal of the ACM **47**(3) (2000) 506–530
13. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation **75**(2) (1987) 87–106
14. Denis, F., Esposito, Y., Habrard, A.: Learning rational stochastic languages. In: Proceedings of COLT'06. (2006) to appear.
15. Denis, F., Esposito, Y.: Rational stochastic language. Technical report, LIF - Université de Provence (2006)